10

Packet Tracer and IoT

LEARNING OBJECTIVES

After studying this chapter, students will be able to:

- explain the basics of Packet Tracer and Blockly programming language.
- design simple IoT projects in Packet Tracer.

10.1 IoT and Packet Tracer

This chapter guides you on how to make simple IoT projects in Packet Tracer. Packet Tracer is a visual cross-platform tool (can be run on Linux, Windows, Android, and also macOS) developed by Cisco Systems, which assists in understanding the concepts of computer networking through simulations [1, 2]. It follows the procedure of simple drag and drop method to add/remove network (including IoT) devices of all kinds. In addition, it allows programming in a variety of languages, i.e. JavaScript, Python, and Blockly, to allow a student to perform simulations in simple ways. The IoT project examples developed in Packet Tracer using Blockly language will be helpful for students to understand the basics of Things connectivity over the Internet. Real-life things embedded with electronic devices are able to communicate each other. IoT is an extension of the Internet, that with an aim of providing interconnectivity among our daily use objects and things over the Internet. Blockly is a programming language, which lets you create your own program in a much better and easier way using Packet Tracer.

Major components of Packet Tracer Integrated Development Environment (IDE) are highlighted in Figure 10.1.

You can select and drag any IoT device (as shown in Figure 10.2) into the workspace using the *Device-Type Selection Box* and *Device-Specific Selection Box.*

Enabling the Internet of Things: Fundamentals, Design, and Applications, First Edition.
Muhammad Azhar Iqbal, Sajjad Hussain, Huanlai Xing, and Muhammad Ali Imran.
© 2021 John Wiley & Sons Ltd. Published 2021 by John Wiley & Sons Ltd.

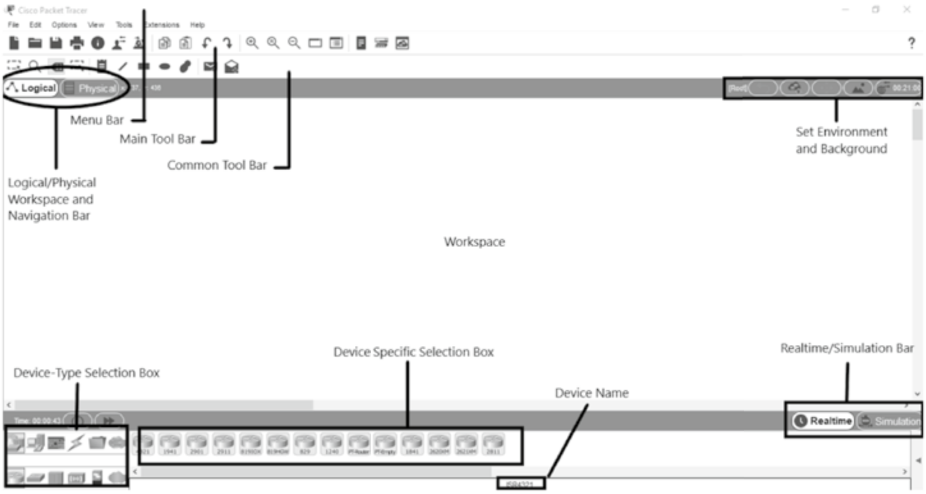

Cisco Packet Tracer
File Edit Options View Tools Extensions Help
Logical
Physical
Menu Bar
Main Tool Bar
Common Tool Bar
Logical/Physical Workspace and Navigation Bar
Set Environment and Background
Workspace
Device-Type Selection Box
Device Specific Selection Box
Device Name
Realtime/Simulation Bar
Realtime
Simulation

Figure 10.1 The Cisco Packet Tracer user interface.

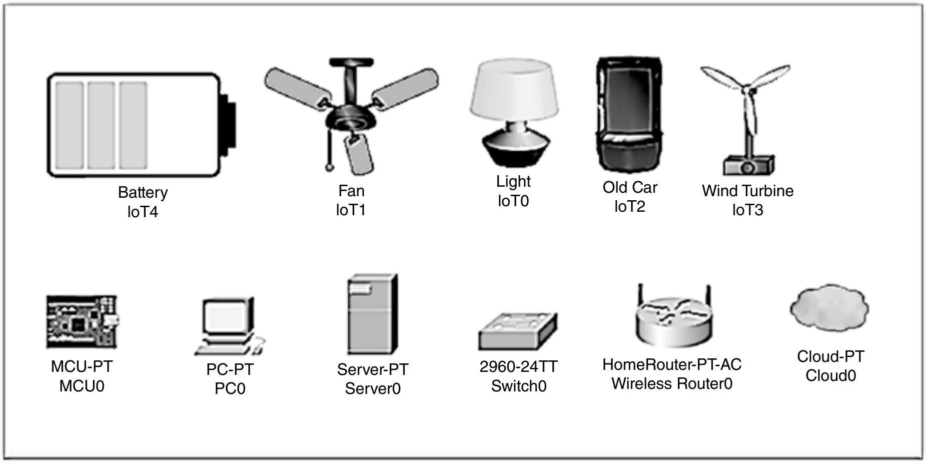


Figure 10.2 Few IoT devices available in Cisco Packet Tracer.

10.2 Packet Tracer Programming Environment

By *double-clicking* on *any of the devices* (already dragged in workspace), a window appears as shown in Figure 10.3. For performing any programming, click on **Programming** tab. Packet Tracer supports three programming languages such as Python, JavaScript, and Visual (Blockly). In order to start new project or to select programming language, click on **New** button.

By clicking on the **New** button, **Create Project** window will appear (see Figure 10.4), which shows text field to enter the project **Name** and to select **template**. From the drop-down menu, here you can see three basic programming languages JavaScript, Python, and Visual as shown in Figure 10.5. In this chapter, we have used Visual programming (another name for Blockly Programming) because it is easy to understand at novice level.

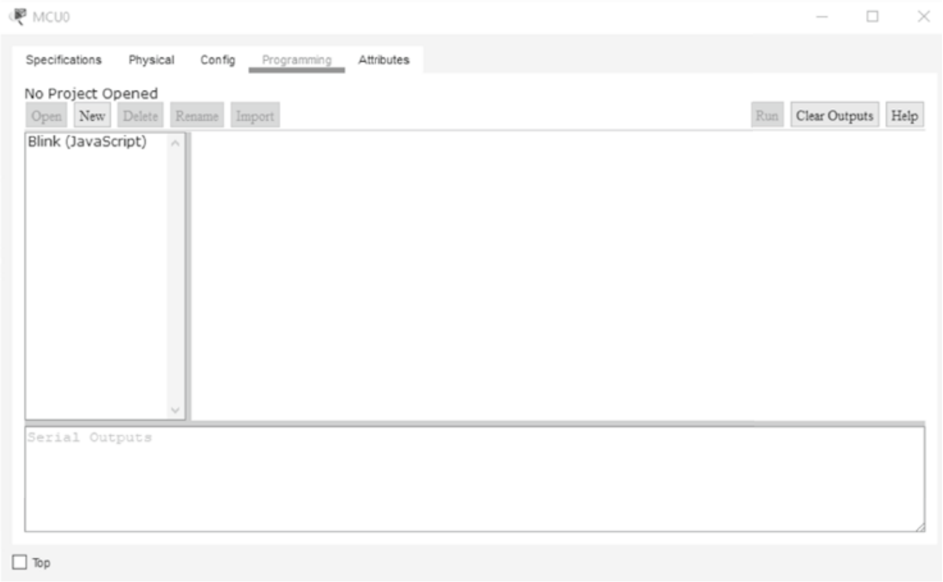
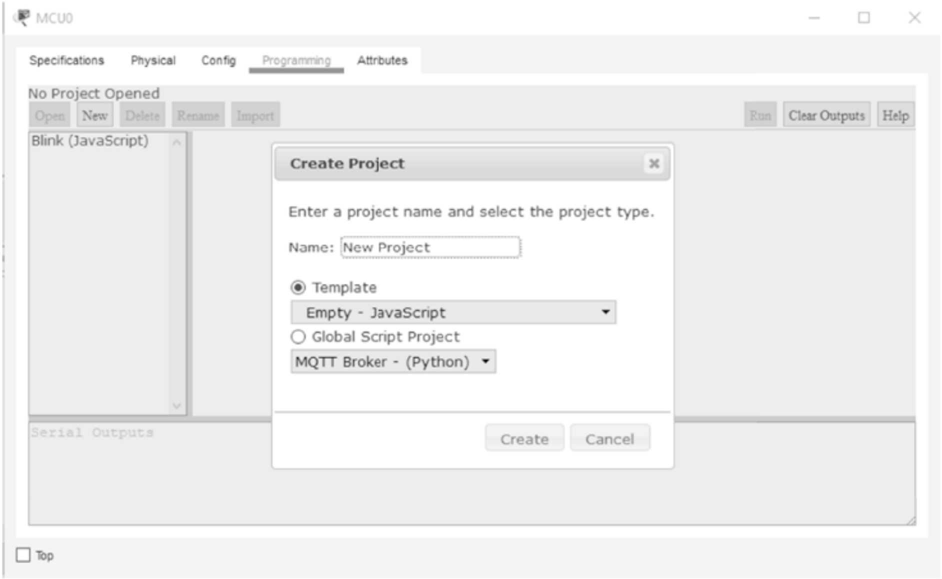


Figure 10.3 Device selection window in Cisco Packet Tracer.



Enter a project name and select the project type.

Name: New Project

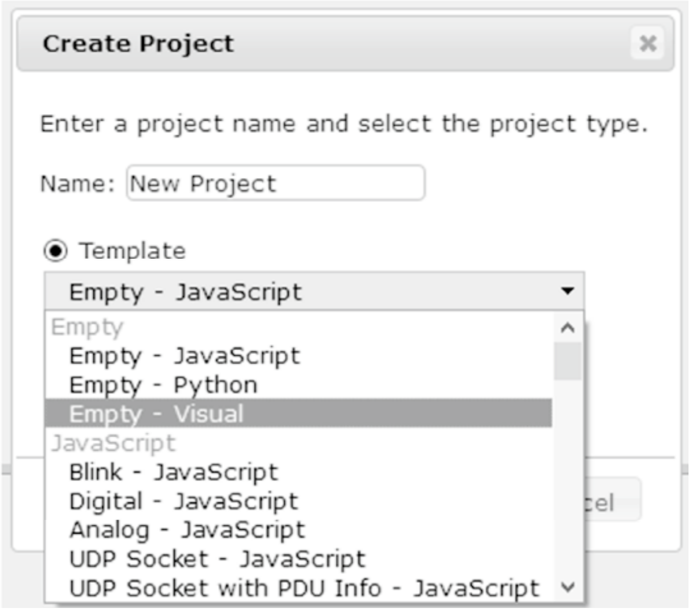
● Template

Empty - JavaScript

○ Global Script Project

MQTT Broker - (Python)

Figure 10.4 Create Project window in Cisco Packet Tracer.



Create Project

Enter a project name and select the project type.

Name: New Project

● Template

Empty - JavaScript

Figure 10.5 Selection of programming languages in Cisco Packet Tracer.

10.3 Visual (Blockly) Programming Language

Visual or Blockly programming language provides visual coding blocks to make the program easier by connecting them together like puzzles. In this section, you will learn about the different blocks available in Packet Tracer, i.e. the names, reason for coloring, and usage. The goal is to get familiar with few blocks and terms, which will be used frequently in the next sections. You can come back to this section anytime and revise their use and recall their purpose.

Blockly editor consists of workspace and toolbox, which consists of many blocks (including functions, arithmetic operations, variables, networking blocks, etc.) in different colors and shapes. Users can drag blocks and generate code easily by rearranging different blocks. Figure 10.6 indicates programming interface step by step: start by selecting ① tab. After giving the project a name as discussed earlier, you can see the project name in ② along with its programming language (Visual in our case). To start programming, double-click the main file ③; another tab ④ appears showing multiple options. We can select any of the tools from here ④ to drag into workspace and later Run the Program ⑥. The output will then show in ⑤ and the execution result will be displayed on the main workspace. Packet Tracer comprises of many blocks for implementing code in a much easier way. To quickly overview all blocks, try different options from panel ④.

Figure It Out!

In *Program*, explore block palette, which comprises of different blocks of functions, variables, maths, logic, loops, etc. and other options such as pin access, networking, TCP, and many more. Blocks are of different colors to differentiate them from each other.

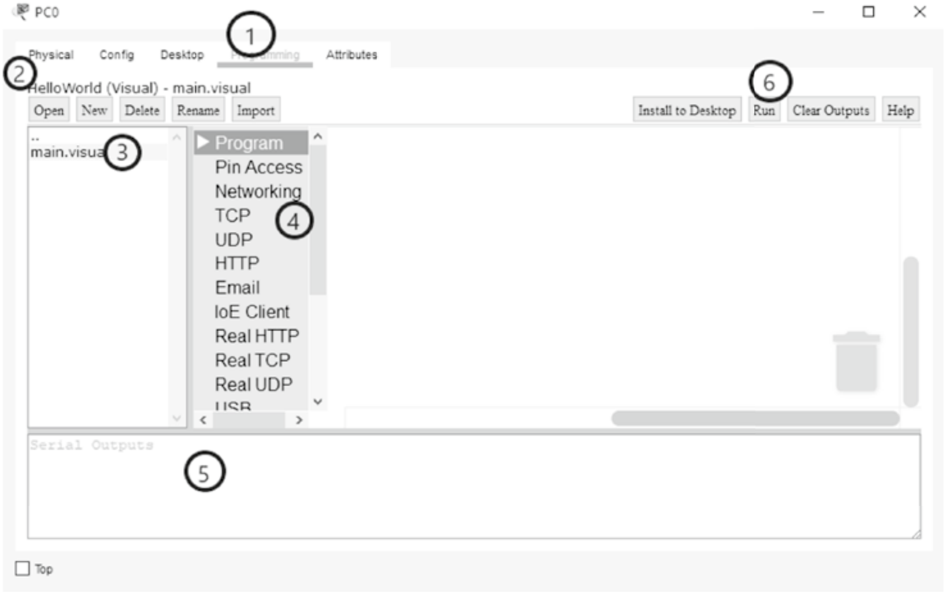


Figure 10.6 Cisco Packet Tracer programming interface.

The Program block contains subsections of blocks: Functions, Variables, Logic, Loops, Math, Text, Lists, and Dictionaries as shown in Figure 10.7. As illustrated in Figure 10.8, some blocks have inside cuts, some with outside edges, whereas some blocks have both inside and outside cuts. Few blocks have inside puzzle-shape marks, indicating that these blocks require another block to connect with. Block alone has no meaning; every block must connect with any of the blocks in order to execute. The color of the blocks indicates the type of data they return. For example, function blocks are in purple color, all type of variables are in pink-reddish color, and so on. Thereafter, complex and lengthy code can be easily debugged with the type of color they have. Other than Program block type, there are 17 kinds of blocks (i.e. Pin Access, Network, Transmission Control Protocol [TCP], UDP, HTTP, Email, Internet of Everything [IoE] Client, Real HTTP, Real TCP, Real UDP, USB, Bluetooth, File System, Physical, Environment, Workspace, and GUI) are available in Packet Tracer (version 7.2.1.0.218) as shown in Figure 10.9.

10.3.1 Hello World Program

To start the first program *"Hello World,"* click on the **Program** tab; select the **Textfield**. A tab appears that represents different blocks from text field. There are blocks available in square and in oval shape. Click on the **print** block as shown in Figure 10.10. It will automatically appear in your workspace. You can observe that this print block have inner blended edge at the top, puzzle mark on right, outer edge at the bottom, and is in light green color. This is because print block requires something to print and that block requires

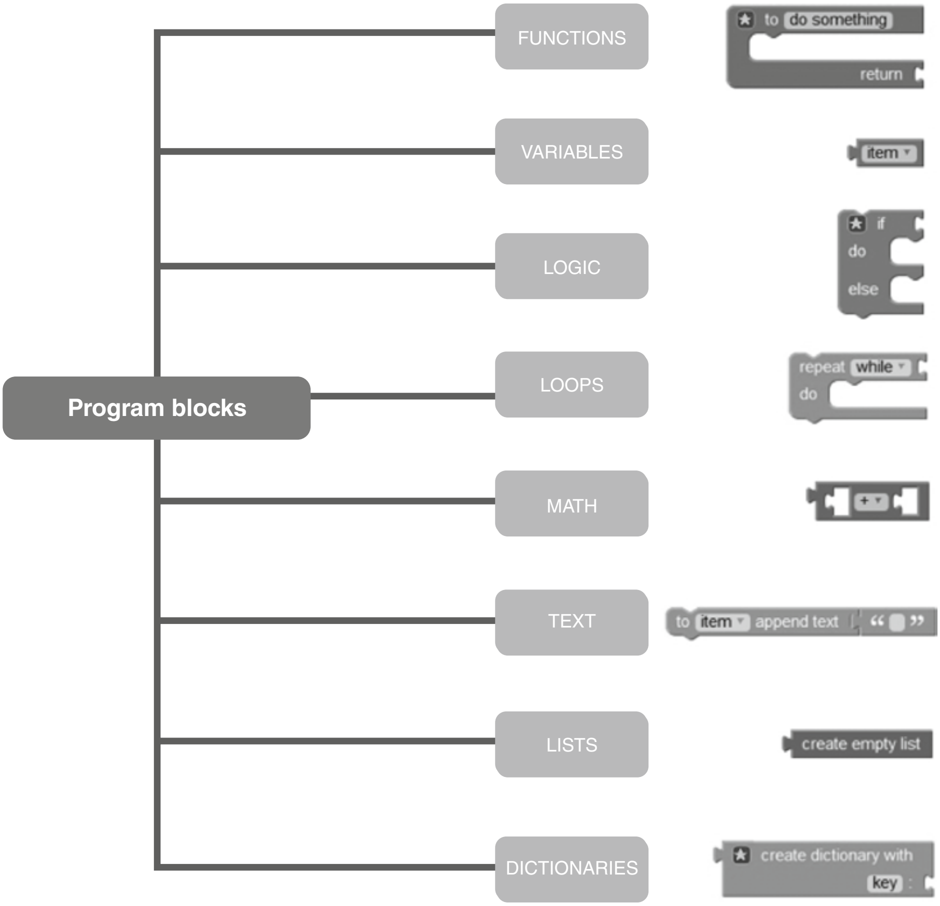


Figure 10.7 Types of program blocks.

another block to connect with it. Just like puzzles require another puzzle to link with to display a picture, similarly blocks in Blockly programming required other blocks to connect with to execute something.

After that click on the **Text** field again and select the very first block having **double quotes** on it, which indicates a string type data can be placed in it. Click the block and write text Hello World in it. Using a mouse to hover any block helps you getting tooltip, which can provide guidance related to all blocks as shown in Figure 10.11. Drag and join the two blocks together as shown in Figure 10.12. Here, yellow borderline indicates that the block can be fixed with another block.

Now you need to execute the program. For this, click on the **Run** button as shown in Figure 10.6 at ⑥. The program will start executing without errors and can be seen under output window ⑤ see Figure 10.13.

To stop executing the program, click the Stop button ⑥ shown in Figure 10.6.

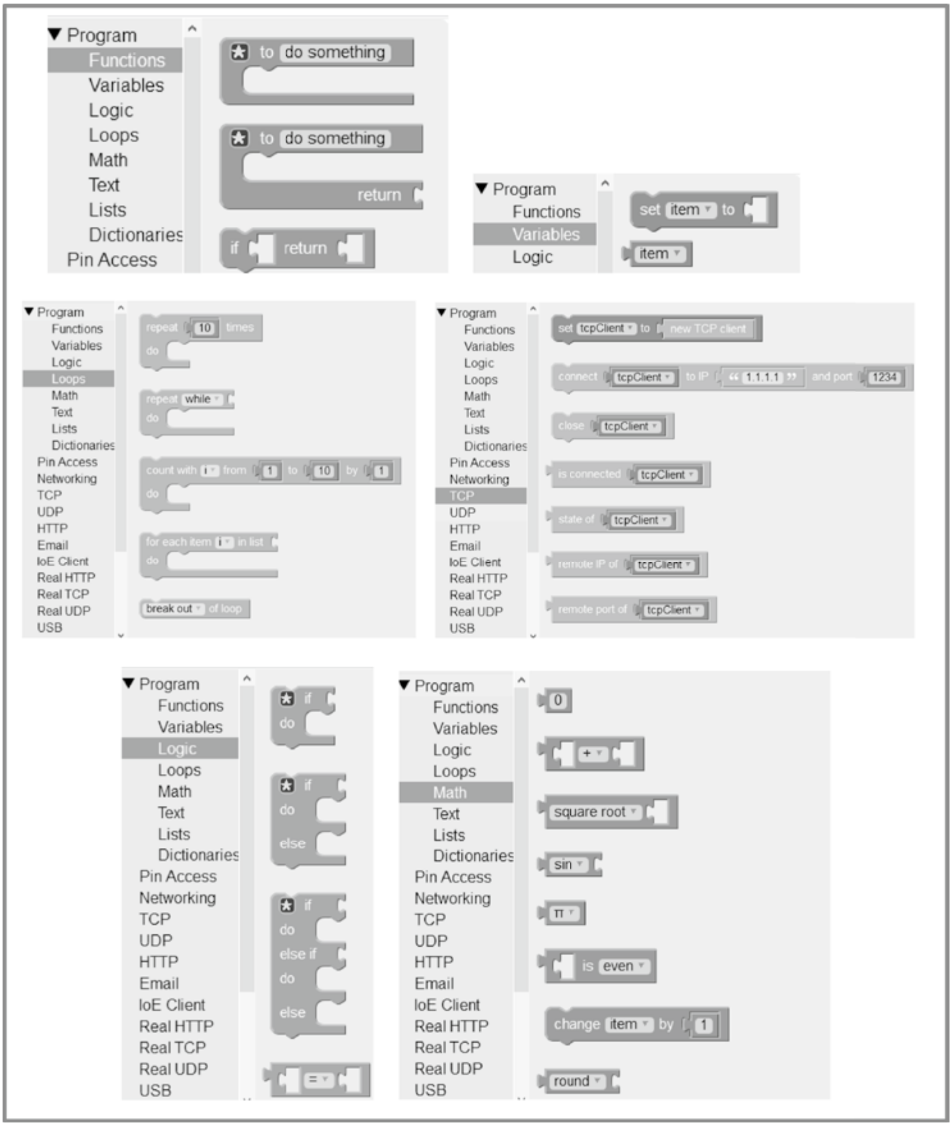


Figure 10.8 Programming blocks available in Cisco Packet Tracer.

10.4 Simple Smart Light Project

As you already know the programming interface and its environmental tools at this stage, you are now ready to use these tools in a new way by developing a new program. You have also learned different blocks and from now onward you can easily use them. In this section, you will learn:

- How to use workspace in Packet Tracer
- How to build blocks together to create a Blockly program

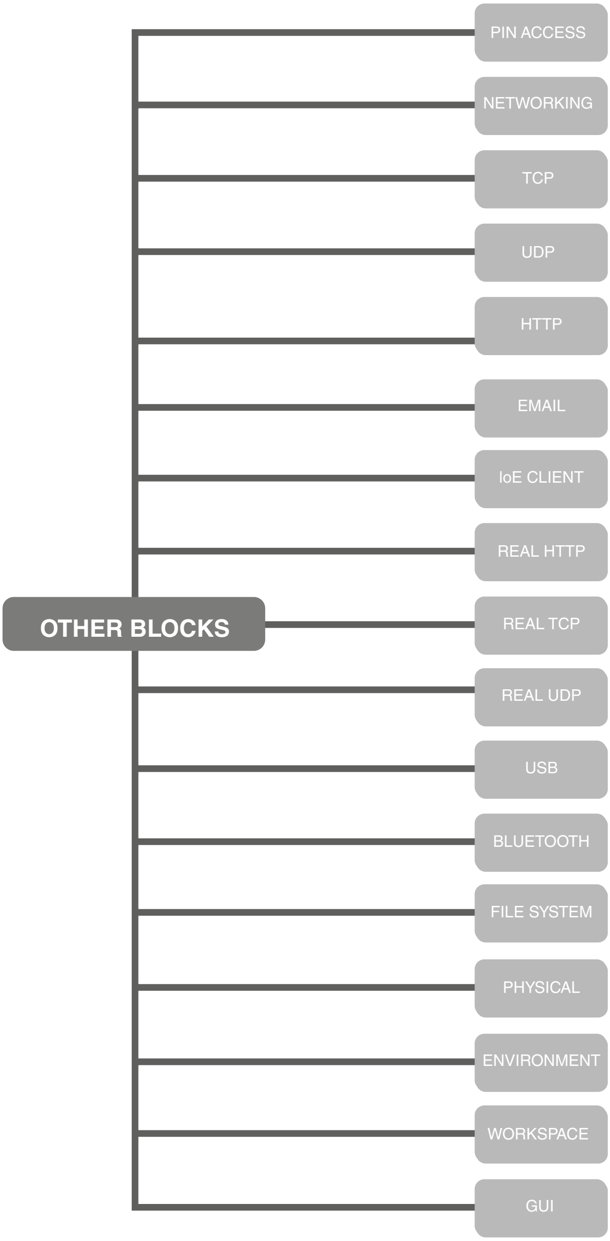


Figure 10.9 Blocks available in Cisco Packet Tracer.

- How to execute a smart light program in a realistic way
- How Blockly makes programming easy
- How to use loops

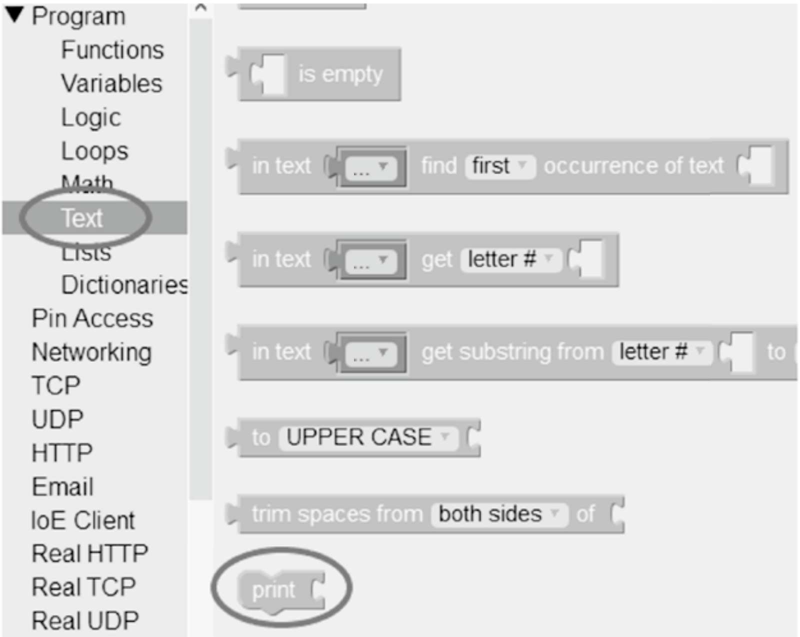


Figure 10.10 Blocks to print and format text.

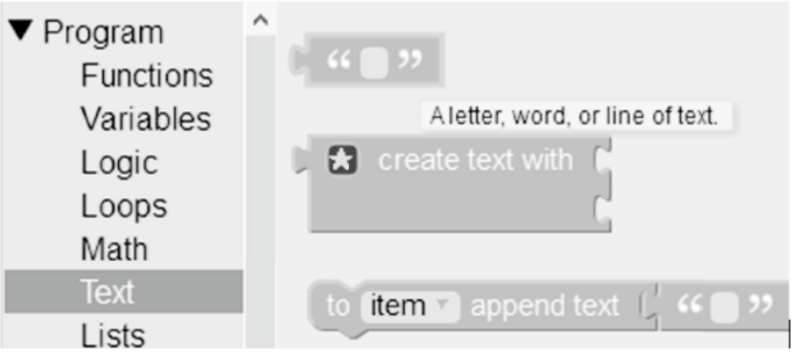
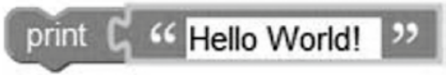


Figure 10.11 Text field block with tool tip.

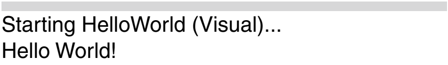
The word *smart* means to convert ordinary things into being smart, which can take decisions based on certain circumstances. Being a *smart device* means it must have underlying technology, power computation, some backend programming, or communication features. This section will help you create a smart light, which will be controlled with the help of a power button that covers these two basic features:

- When the power button turns ON, the lamp is turned on and the light starts blinking.
- When the power button turns OFF, the lamp is turned off.



```
print “ Hello World! ”
```

Figure 10.12 Hello World block.



```
Starting HelloWorld (Visual)...
Hello World!
```

Figure 10.13 Output window.

To implement this, start by executing Packet Tracer. To create a new project, click from the menu toolbar **File → New**. A blank workspace will appear. (See Section 10.1 for the details of all toolbar and programming environment interface.)

10.4.1 Adding Devices to Workspace

A list of all devices can be seen at the left-bottom corner. There you will see multiple devices of various categories. Select the **End Devices** section and from this select **Home** icon as shown in Figure 10.14. You can also apply shortcut keys by pressing **CTRL + ALT + V** (End Devices) or **CTRL + ALT + H** (directly selecting Home section).

From there, you will see a variety of home icons used in our daily lives, such as air conditioners and home appliances such as coffee makers, batteries, fans, lights, doors, and so on. Select the **Light** device and click on workspace; you will note a plus sign appear on workspace indicating where to drop light in workspace. While selecting any device, you will notice a cancel mark appear on that specific device, in case you do not want to drag it on workspace. Simply cancel or delete it by pressing delete key from the keyboard.

Now, you have dragged a light device into the workspace, you will notice that a light device has a label with it IoT0. By default, all devices have labels according to their category. This device is from the IoT, also called Smart Devices and number as 0. The more you drag, the label would be like IoT1, IoT2, and so on. To rename a label click on the label (IoT0) and rename it as shown in Figure 10.15.

Now, delete the extra lights as we only need a single light to deal with. The next device we need is the power button, which has two options ON and OFF. From now onwards we name power button to be as *Rocker Switch* button. To drag a rocker switch into workspace, click the **Components** section **(CTRL + ALT + B)** and then select the **Sensors (CTRL + ALT + X)** option as shown in Figure 10.16, and select the **Rocker Switch** from the list having label 1 and 0.

If you click on the device present on a workspace you will see its features and all the related information in detail. For example, in case of light, click on a light device; a window will appear that will define light, its features, and the role it played. Similarly, if the rocker switch is pressed, you will see its relevant details in the **Specifications** tab as shown in Figure 10.17.

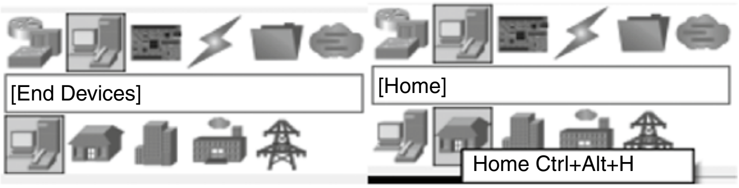


Figure 10.14 Selection of devices.

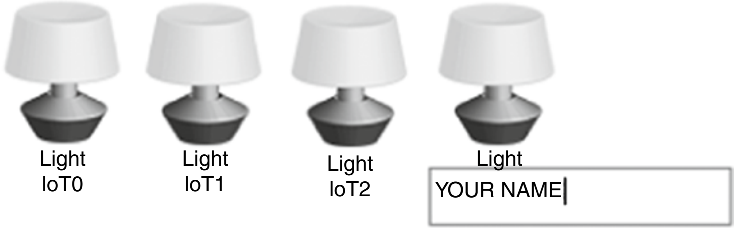


Figure 10.15 Rename a device.

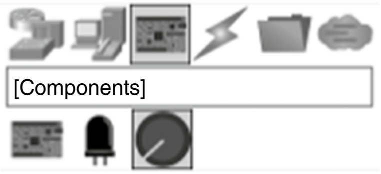


Figure 10.16 Selecting rocker switch from the components.

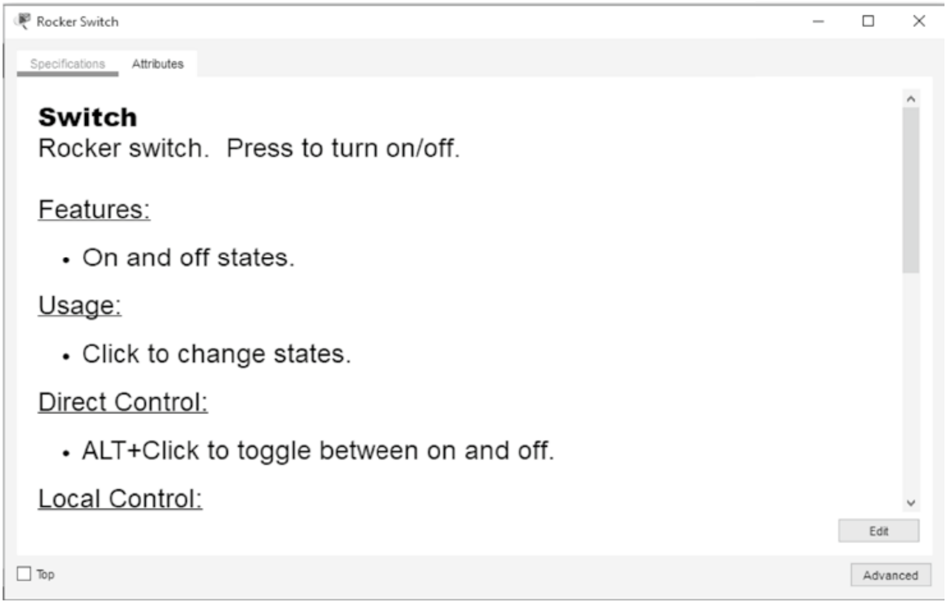
To see every device's role and animation, press **ALT + CLICK** together to toggle between device options. A light can be turned on and off. In our case, a light can be dim and high. Similarly, fan speed can be slow or high. To observe such behavior, press **ALT + CLICK** on lamp; you will see that the first click makes lamp light *DIM ON*, second click makes lamp light *HIGH ON*, and third click turns the lamp *OFF* as shown in Figure 10.18.

Figure It Out!

Try these toggle options against multiple devices such as fan, switches, coffee maker, portable music player, and so on and find out those devices who don't have such options.

Now, we have two devices on the workspace, light and rocker

switch working individually. The problem here is we want to control light through switch. If the switch is ON, the light turns on automatically, whereas, if the switch is



Switch
Rocker switch. Press to turn on/off.

Features:

- On and off states.

Usage:

- Click to change states.

Direct Control:

- ALT+Click to toggle between on and off.

Local Control:

Figure 10.17 Description for rocket switch device.

turned off, the light goes off smartly. For this, there comes the programming part on a third device, which can communicate with light as well as switch. From the devices, select **Board (CTRL + ALT + B)** devices from the **Components** section. There you will see two types of boards – Microcontroller (MCU) board and Single Boarded Computers (SBC) Board. We will deal with **MCU Board** . Drag the board in the workspace (Figure 10.19).

MCU is a microcontroller board that supports multiple programming languages used mainly for controlling other devices through programming.

10.4.2 Connecting Devices

The three devices on the workspace are still not connected with each other. For this purpose, we need a cable that connects all the three devices together. Select the **Connections** tab **(CTRL + ALT + O)** as shown in Figure 10.20 and select **IoT Custom Cable** from the list.

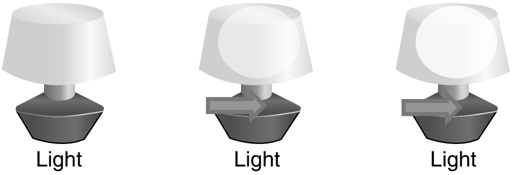


Figure 10.18 States of light (OFF, DIM, HIGH).

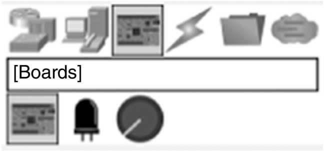


Figure 10.19 Selecting the MCU board from the components.

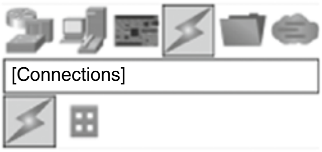


Figure 10.20 Selecting cables from the connections.

The mouse icon changes, which allow you to click on the rocker switch device and select the **D0** port. Now the cable requires the other device to connect with. Click on MCU board and select **D0** port as shown in Figure 10.21. The two devices are now connected.

Now, repeat the procedure for the light device. Select the IoT custom cable again and click on the light **D0** port with MCU board **D1** port. As D0 port is already busy with the rocker switch button, the final look of the connections will be like the one shown in Figure 10.22.

That's not enough. The functionality is not yet complete. To verify ALT + CLICK on switch button which toggles switch states but light is not affected. Here starts the programming part for Blockly. Now, double-click the MCU board; a window will appear, which allows you to switch between Specifications, Physical, Config, Programming, and Attributes tab. Select the programming tab as shown previously in Figure 10.3. Click on the New

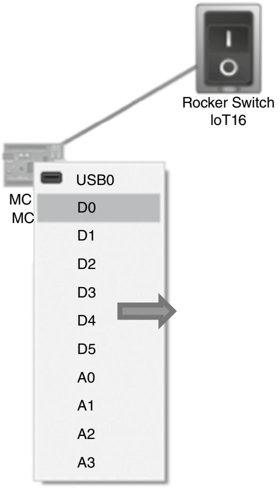


Figure 10.21 Connecting devices through connection cable in Cisco Packet Tracer.

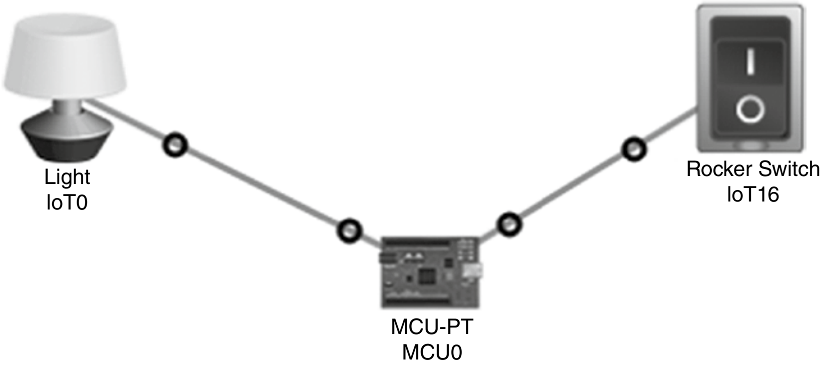
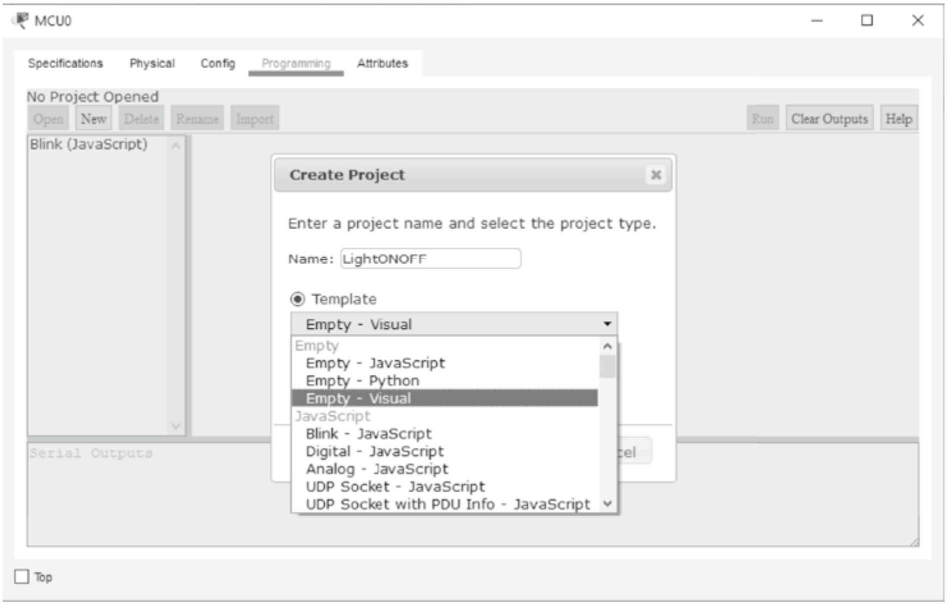


Figure 10.22 Connecting all the three devices together.

button to start creating new project. Give your project a name. The Template option is selected by default. From the drop-down menu of the Template section, select the Empty - Visual option (shown in Figure 10.23) and click on Create button. From the drop-down menu, here you can see three basic programming languages JavaScript, Python, and Visual as shown in Figure 10.5. Our focus will be on Visual, another name for Blockly Programming.

On the left side of the panel, you will see your project name and main.visual. Double-click main.visual and the block panel will appear as shown in Figure 10.6, where the interface is explained in detail.



No Project Opened

Create Project

Enter a project name and select the project type.

Name: LightONOFF

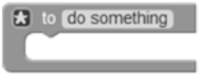
● Template

Empty - Visual

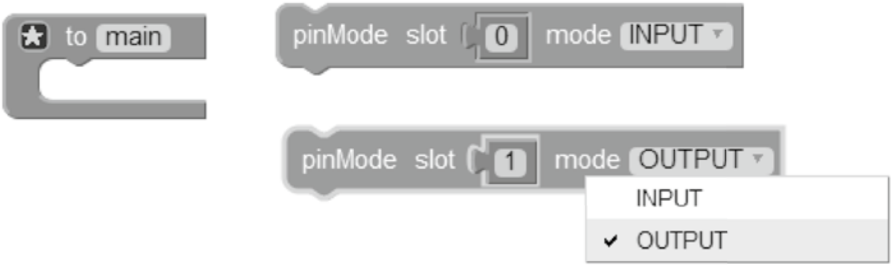
Figure 10.23 Creating new project.

10.4.3 Using Program Blocks and Pin Access

In order to create a program that requires mathematical computation or requires some sort of logical implementation or repetition of any task or in any case just to create variables to handle the programming environment, we use **program** blocks from the block palette. You will find out how in this section.

Every program must contain the main function, which allows program to start. For this purpose, from the program tab under function option, select  to do something block. This will appear on the workspace of programming interface window. Always assign functions name that reflects its nature; therefore, rename and do something with main like this to main to make function name as user friendly. You can drag the blocks anywhere on the workspace and adjust accordingly. This main function is defined now and is ready to call. To call a function, the new block of main is main added automatically in the Function section. Add this block anywhere into workspace. So, whatever blocks we put into this function main to main will be called by this main.

From the Pin Access section, select pinMode slot 0 mode INPUT. As you have observed this block has inside cut at the top and outside edge from the bottom, which indicates that this block requires to connect with two more blocks to show its functionality, which matches its edges accordingly. Also, pinMode has two parameters by default, slot 0 and mode INPUT. You can change these two settings according to the cable connections you made earlier in this chapter. Now repeat this process again by selecting pinMode slot 0 mode INPUT again from Pin Access section into workspace. However, this time change slot 0 to 1 and from the drop-down menu change mode to OUTPUT. Your workspace would look like the one shown in Figure 10.24.



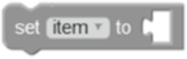
to main

pinMode slot 0 mode INPUT

pinMode slot 1 mode OUTPUT

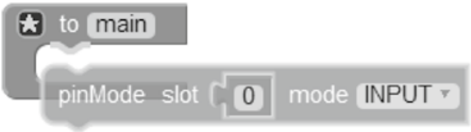
Figure 10.24 Settings pinMode parameters.

Now, what is this all about? As shown in Figure 10.22, the MCU board has two cables attached, one with rocker switch connected with port D0 and the other with light port D1. So, we have to configure two pins against two cables attached. We want signals to come from rocker switch as INPUT to MCU board and after some operations pass those signals to light as OUTPUT from MCU board. Remember, we are in MCU board programming section; therefore, input and output modes will set according to MCU board. That's why we set pinMode D0 as Input (Don't write D with 0) for rocker switch device cable and pinMode 1 as Output for light device. Drag the two blocks one by one into main function block. The yellow line indicates that the two blocks can join together as their cut marks matched as shown in Figure 10.25. Repeat the same with the other block as shown in Figure 10.26.

We need now to get signals from rocker switch in a way that if the signal indicates 1 light should turn on and if the signal indicates 0, the light should turn off. Therefore, it indicates that rocker switch deals with digital signals also explained in the Specification tab of rocker switch. In order to read signals digitally from rocker switch, select from Pin Access section. Set slot to 0 (already set by default). This will return the signal from port D0 and will give its value, which should be stored somewhere. In programming, we used to store values in variables. Thus, a new variable must be created first, which can store the resultant signal value. To create a new variable, select  block from Variable section and rename it or simply create New variable as shown in Figure 10.27 as follows:

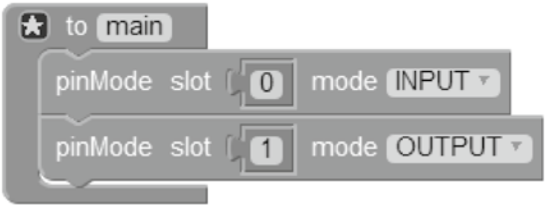
Rename a variable to be as *switchValue* or a name of your choice and press **OK**. You will notice the change in the variable palette with the addition of two new blocks as shown in Figure 10.28.

Set *switchValue* to what? Whenever we create a variable at the start, we used to assign its value to 0 to avoid garbage results. Therefore, from **Math** section, add and attach it



to main
pinMode slot 0 mode INPUT

Figure 10.25 Connecting blocks together. Source: Cisco.



to main
pinMode slot 0 mode INPUT
pinMode slot 1 mode OUTPUT

Figure 10.26 Inserting blocks into function.

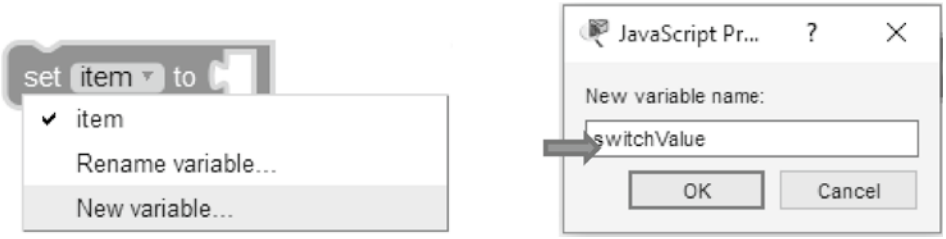


Figure 10.27 Creating a new variable.

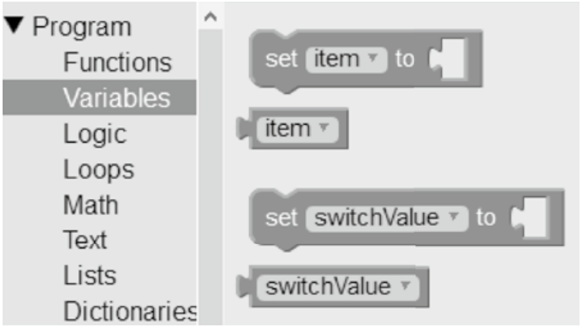
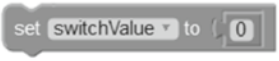
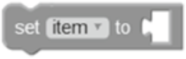
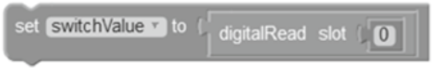
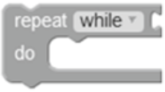
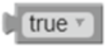
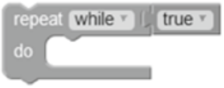
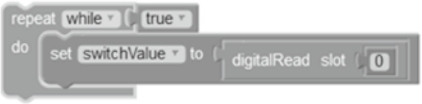
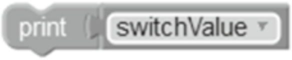
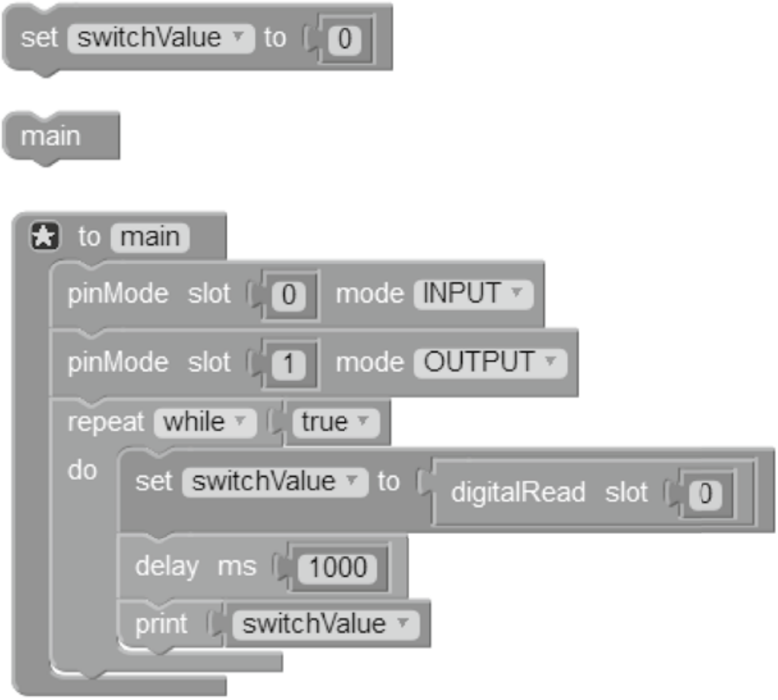


Figure 10.28 Addition of new variables.

to the variable added before like . Initially, the *switchValue* is set to 0. As soon as we get value from digital signal, the value needs to be updated; therefore, from the **variable** section, add  again and attach it with the brown block (we added previously) . This means whenever Pin 0 (cable D0) sends signal, it will be read and the value is assigned to a defined variable.

Now, whenever we press the rocker switch sensor button to be ON, the light should turn on and off accordingly. This process must be repeated again and again. Therefore, here we need a loop, which constantly checks the values and repeats this process until the condition is satisfied. For this, select while loop  from Loops section. The loop required to execute until the condition is true; therefore, from the Logic section add block  into workspace and attach in front of the while loop . Drag the digitalRead blocks into loop . Now the values from the sensors (rocker switch) will be checked constantly without any delay, which creates a program to be stuck sometimes. Therefore, adding delay to the program resolves this issue. To add delay of 1000 ms, go to Pin Access and drag it delay ms 1000. Add this block into the loop after variable blocks. To test whether the program is reading digital values or not, we need to display something on the output window. For this purpose, add print from the Text section. This block demands something to print. You can add any string with this or add a variable to be print. Let's check the signal value to be displayed. Thus, from the Variable section, add variable and attach the two blocks together  print switchValue. Your programming environment workspace will look like the one shown in Figure 10.29.

To execute the program, click on the Run button. You will start getting 0 output in the *Output* window as shown in Figure 10.30, which displays the project name and its output. Now, go back to your main workspace of Packet Tracer where the sensors are placed. **ALT + CLICK** on the rocker switch sensor once. It will change its state from 0 to 1 and a

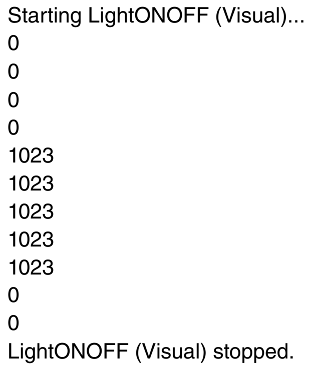


```
set switchValue to 0

main

to main
  pinMode slot 0 mode INPUT
  pinMode slot 1 mode OUTPUT
  repeat while true
  do
    set switchValue to digitalRead slot 0
    delay ms 1000
    print switchValue
```

Figure 10.29 Light program test mode.



```
Starting LightONOFF (Visual)...
0
0
0
0
1023
1023
1023
1023
1023
0
0
LightONOFF (Visual) stopped.
```

Figure 10.30 Light program output window.

green light turns on at the sensor . Now go back to output window; you will start getting signal values constantly with the delay of 1 s which is 1023 as shown in Figure 10.30. Toggle between the ON and OFF button of the sensor to get values 1023 and 0. To stop the execution click on Stop and to clear output window click Clear Outputs.

You can also print a text that defines the current situation instead of values 0 and 1023. Like when you press OFF, the output would be like "button off," and when you press ON, output would be like "button on." For this add " " from the Text section and write the text "button off." Replace the block print (switchValue ▾ with the text block print (" button off " . In this case, you will need to handle if-else statements too (we will discuss in the coming sections).

The functionality of handling the light is not yet complete. However, the reading commands from the sensor is completed as we are getting proper signals from the sensor, reading and storing it in a proper variable. Now, our aim is whenever we read 0 signal from the sensor, the light remains off; however, the light turns on as soon as we get signal 1023 from the sensor. To implement this functionality, we have to write some commands for light. For writing commands, we have *customWrite* blocks available in the palette. To add, go to **Pin Access** section, find the block customWrite slot (0 value (and add it to the programming workspace. Change the slot from 0 to 1. As the connected port from MCU board to light is 1 as shown in Figure 10.31, we need to write the commands on port 1. Set its value to 2 by adding a number block 0 from **Math** section like this customWrite slot (1 value (2 . You can get port number information by just hovering the black circles on the cable as shown in Figure 10.31.

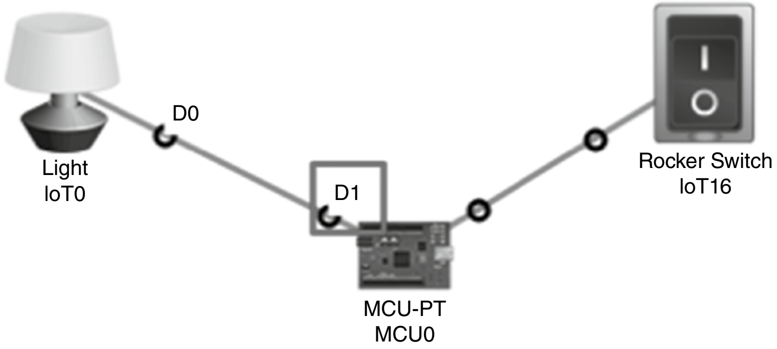
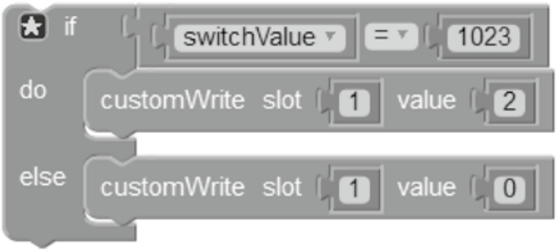


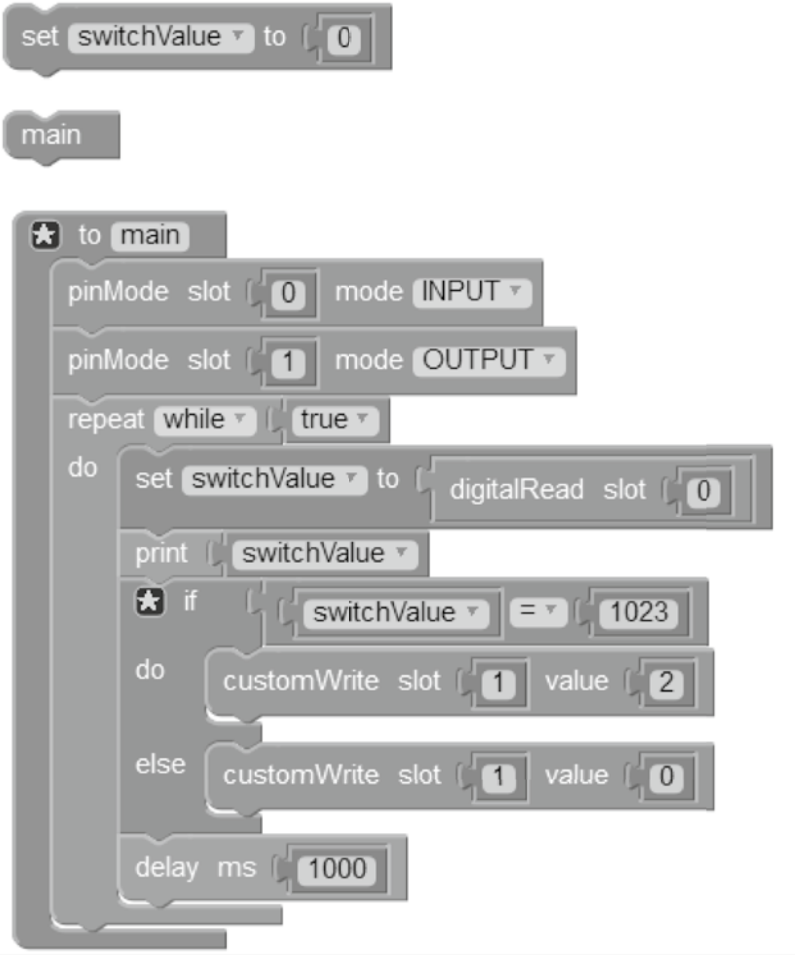
Figure 10.31 Writing D1 port.

There arise two situations: (i) if the signal is 0 then the light remains off and (ii) else if the signal is 1023 the light turns on. This means we need if-else block, which will handle the situation. For this, go to **Logic** section, and add block . Now *if* statement follows what *condition*? Yes, we need here a variable to compare the signal reading with; that's why we store the signal values in a variable named *switchValue*; thus if the switchValue is equal to zero, light will be off else on. Therefore, add variable *switchValue* in if statement by adding block from the **Logic** section. Join the block to be first parameter and add numeric value 1023 to be second from the **Math** section. The block will look like . Join the block with the *if* statement of the block. Now, if this statement is satisfied, the if block executes the *do* statements. Join the block into if block. What *else* part will do? Surely, the else block makes the light off; therefore add another and set the value to be as 0. The if-else block will look like the one shown in Figure 10.32 and the complete program is shown in Figure 10.33.



```
if   switchValue = 1023
do   customWrite slot 1 value 2
else customWrite slot 1 value 0
```

Figure 10.32 If-else block statements.



```
set switchValue to 0

main

to main
  pinMode slot 0 mode INPUT
  pinMode slot 1 mode OUTPUT
  repeat while true
  do
    set switchValue to digitalRead slot 0
    print switchValue
    if switchValue = 1023
    do
      customWrite slot 1 value 2
    else
      customWrite slot 1 value 0
    delay ms 1000
```

Figure 10.33 Complete flow for the light program.

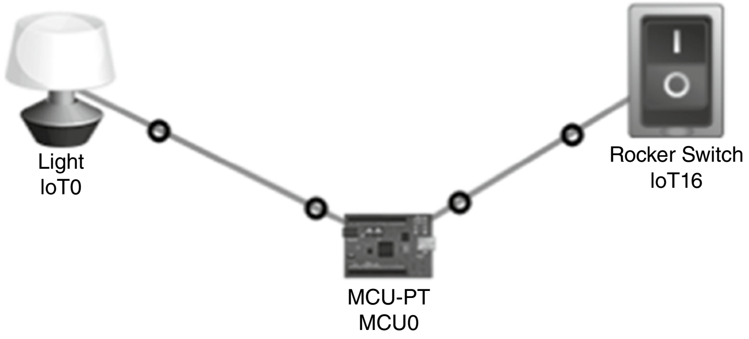


Figure 10.34 Light program in execution mode.

Test the program by clicking on Run tab and toggle between the sensor switch as shown in Figure 10.34. Now, the light turns on as the sensor gives readings and off otherwise. The light in Packet Tracer has three states: light on but DIM = 1, light on but HIGH = 2, and light off = 0. To implement these three states, the slots' values are required to be set accordingly.

References

1 Janitor, J., Jakab, F., and Kniewald, K. (2010). Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer. In: *IEEE 2010 Sixth International Conference on Networking and Services*, 351–355. IEEE.

2 Jesin, A. (2014). *Packet Tracer Network Simulator*. Packt Publishing Ltd.